Bi-level Hierarchical Layouts for Photo Libraries: Algorithms for Design Optimization with Quantum Content

Jack Kustanowitz, University of Maryland
Ben Shneiderman, University of Maryland

ABSTRACT

A frequently-used layout for a collection of two-dimensional, fixed aspect-ratio objects, such as photo thumbnails, is the grid, in which rows and columns are configured to match the allowed space. However, in cases where these objects have some group relationship among them, it can be advantageous to show this relationship in the layout, rather than in textual captions. We use an annotated digital photo collection as a case study of an auto-layout technique in which a two-level hierarchy is generated, consisting of a primary, central region with secondary regions (typically 2-12 regions) surrounding it. We show that given specific requirements, this technique is also optimal, in the sense that it will generate the largest size for the objects. Since all objects are the same size we refer to them as quantum content. These algorithms are designed to be real-time, enabling a compelling interactive display as users resize the canvas, or move and resize the primary region. The interactive redisplay also occurs as users add regions or objects to a secondary region.

1. INTRODUCTION

Page layout underwent a revolution with the advent of automatic text-flow algorithms, and writers and publishers now take for granted that text will break correctly at the end of a line, or flow around an image or drawing within a document. Today's web browsers work on this same principle of dynamic flow, and most web pages will reflow the text as the window is resized. If users print a document, properly configured text will re-flow to meet the paper's dimensions.

This paper explores the possibility of doing rapid automatic layout with non-overlapping two-dimensional fixed aspect-ratio objects, such as photos. These objects can appear in a central primary region, or in secondary regions surrounding the primary region. The techniques presented here are especially applicable to photo libraries, but may have applications in chip design, newspaper layout, and even city planning. A compelling aspect of these algorithms is the interactive redisplay as users change the rectangular canvas size and shape, add secondary regions, or add objects to a secondary region. This kind of animated interaction is an important feature for consumer applications such as personal photo management.

2. CASE STUDY: PHOTO LAYOUT

In many commercial photo management tools, such as ACDSee [1], Adobe [2], iPhoto[3], and Picasa [10], photos are presented in a simple grid. Since all photos are the same size we refer to them as quantum content. While these programs allow advanced metadata annotation, such annotation does not carry over to the presentation mode so that users could view photos with different annotations simultaneously, sorted appropriately. Our work extends the design in newer tools such as PhotoMesa[9], which show the benefits of a grid layout, based on directory structure or metadata annotations. Following are several screenshots with accompanying descriptions that illustrate how such a bi-level layout presents interesting possibilities for this domain. More extensive justification and a user study appear in [8].

Family Photo Collection: Figure 1 depicts a family layout, in which photos for each family member have been chosen. In it, similar quantities of photos are in each region, which lends itself to a balanced view. Layouts with reasonably similar quantities should be able to minimize wasted space, and with some user manipulation of the size/position of the primary region, create a layout that is efficient and attractive.

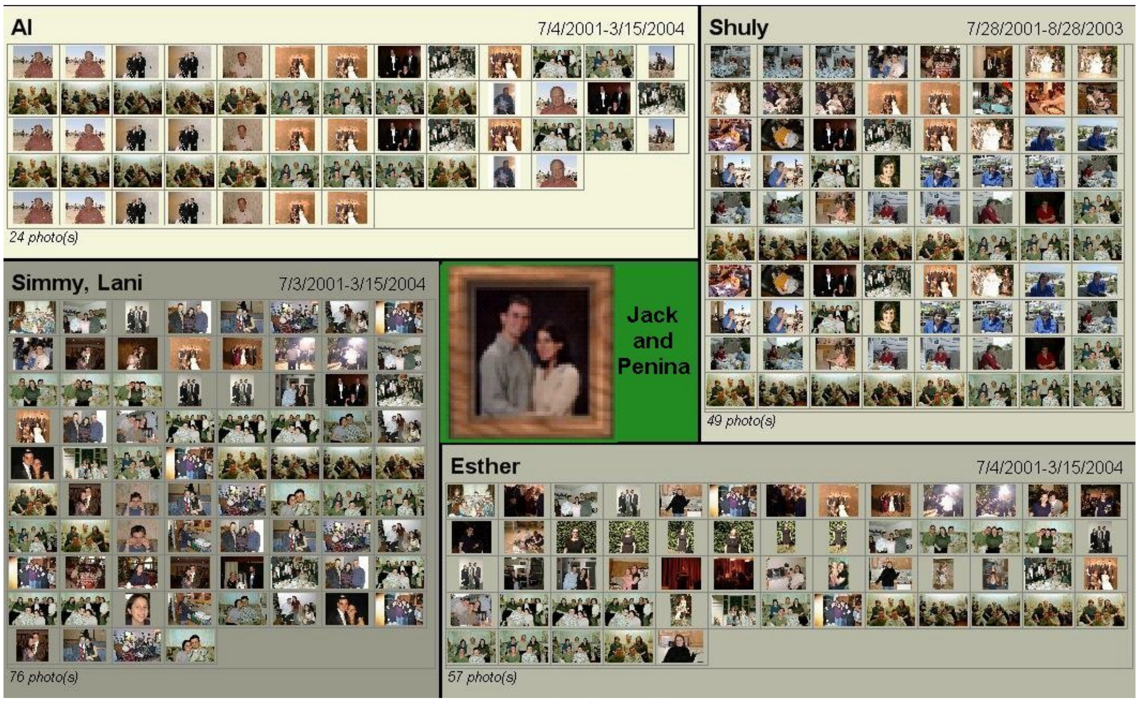


Figure 1: Family photo collection

Vacation Trips: Figure 2 depicts a generated layout of photos taken on a trip to Italy. This is a case in which the ordering is important, so that users can readily find a part of the trip that was towards the beginning, middle, or end. This is also a good example of having more than four regions of photos, along with the need for a dynamic and proper balancing.

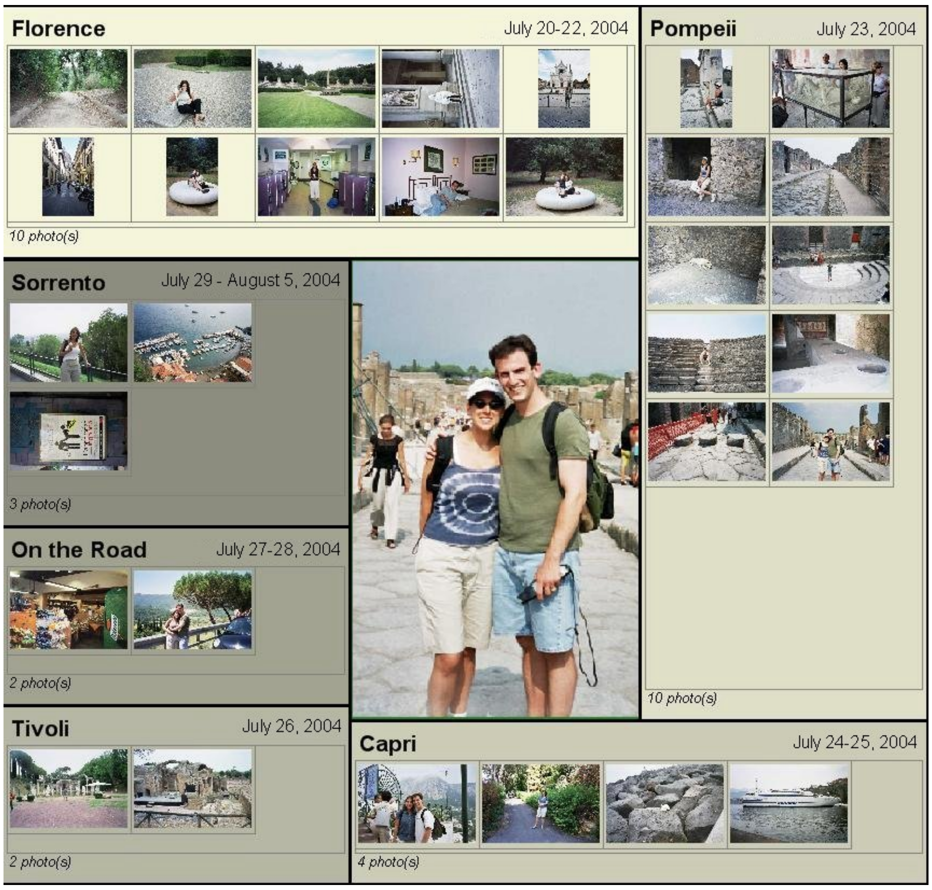
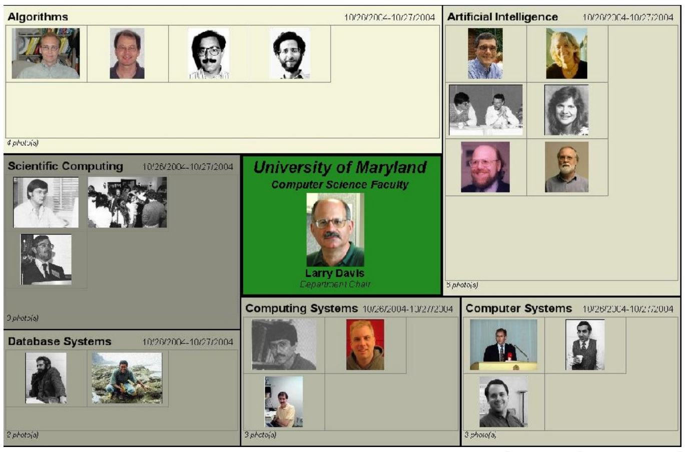


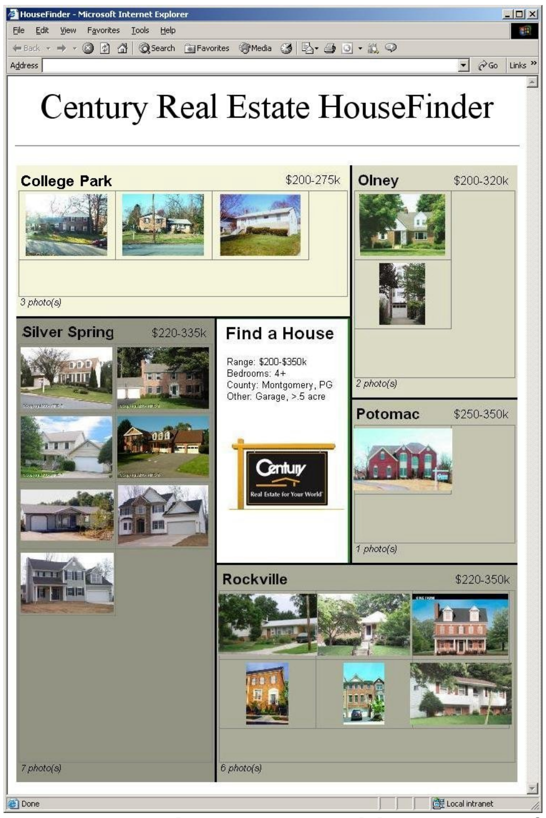
Figure 2: Honeymoon in Italy showing 6 locations during a two week trip

Organization charts: A common grouping is to represent organizational hierarchies, but they do not usually have a sequential layout requirement. Figure 3 shows six regions representing the research areas of the University of Maryland Computer Science department with a photo for faculty members. The primary region shows the Chair of the department. In this case, a layout was chosen that contained blank space in order to provide long enough horizontal space for the titles. As an example, if the primary region were enlarged to the left, the thumbnail size could remain constant, but the text "Scientific Computing" would get truncated.


Algorithms 10/26/2004-10/27/2004
4 photo(s)
Artificial Intelligence 10/26/2004-10/27/2004
5 photo(s)
Scientific Computing 10/26/2004-10/27/2004
3 photo(s)
University of Maryland
Computer Science Faculty
Larry Davis
Department Chair
Database Systems 10/26/2004-10/27/2004
2 photo(s)
Computing Systems 10/26/2004-10/27/2004
3 photo(s)
Computer Systems 10/26/2004-10/27/2004
3 photo(s)
Figure 3: UMD Computer Science Department organization chart with 6 research areas

Real Estate Browser: Figure 4 presents a hypothetical real estate web site, containing a bi-level radial quantum photo layout within a web page. As a user changes the size of the browser window, the size of the layout, regions, and photos all size dynamically to allow for the largest photo size possible given the relative number of photos in each area. In this way, the photo flow resembles text re-flowing on resize.

Should a new house come on the market in Silver Spring, it would fit nicely in the blank space under the houses currently visible. A new house in one of the other regions could push the photo size smaller, or might instead cause the secondary regions to be distributed differently about the primary region, possibly without requiring smaller photos.



Find a House

Range: $200-$350k
Bedrooms: 4+
County: Montgomery, PG
Other: Garage, >.5 acre

Figure 4: Real Estate Browser with 5 communities and the price ranges of selected homes

3. RELATED WORK

Our work extends the work on ordered and quantum treemaps [7, 4] that are applied in PhotoMesa[9]. Treemaps map a hierarchy onto a rectangular region in a space-filling manner, and ordered treemaps add an additional requirement of maintaining order in the layout. Order preserving layouts reduce the effect of rectangles shifting position as the item sizes change. The quantum requirement is useful in photo layouts, in which each rectangle has sub-rectangles of fixed size. The layout algorithms described in this paper add a central (both in importance and in position) rectangle, which can itself be sized and moved around the layout, with the other rectangles rapidly moving and resizing in turn. This bi-level property is useful to give prominence to a certain feature of the layout, be it a graphic, descriptive text, or a central feature of a non-photo application.

The problem of dynamic flow of quantum-sized items around a central rectangle resembles text flow around photos, which can be partially automated [6]. These web page and newspaper-like layouts provide an inspiration for our work, and they share the goal of creating layouts that work well at any size.

While our approach to the interaction of requirements is to define them in such a way as to guarantee a solution, other creative ways to solve problems with interrelated constraints include using Monte Carlo methods or genetic algorithms [11]. Genetic algorithms have been used to solve image placement problems in the context of automated page layout in a digital album, using principles such as balance, spacing, emphasis, and unity [5]. These algorithms produce useful results that are appropriate in cases where the constraints in question are fuzzy or subjective. In our work they are more quantifiable, which allows for the simpler and faster algorithms presented in this paper. A genetic approach to finding a solution with one or more of the constraints removed would be interesting to explore in future work.

4. PROBLEM DEFINITION AND REQUIREMENTS

As we defined the problem, certain requirements became clear. For example, we wanted the user setting of canvas size to remain stable, even if an algorithm could discover an improved layout by reducing/enlarging the canvas dimension. Similarly, we decided that uniform size for all photo thumbnails was an important goal, even if an algorithm could reduce unused space by reducing/enlarging some thumbnails. Future research (Section 7) may relax some of the requirements we have taken for granted to produce interesting variations that may be applicable in other domains.

4.1. Fixed Canvas Size

The canvas size (width x height) may not be changed by the algorithm, and the layout will be determined by this surrounding rectangle, usually the window size chosen by users. As users vary the canvas size (the algorithm runs again with each resize), the layout will change but will always fill the surrounding rectangle exactly, never causing it to grow or shrink under algorithmic control.

Relaxing this requirement would let the canvas size grow if the algorithm determined that a better layout could be achieved if the canvas were 5% wider, for example. While possibly useful, this introduces a feedback loop in which the canvas size determines layout, which in turn determines canvas size. Preserving this requirement allows the algorithm to be deterministic, avoids this non-linearity, and maintains the user’s sense of control.

4.2. Fixed Primary Region Size & Location

The size and location of the primary region is set by users and not modified by the algorithm. If users change the primary region size and location, then the layout will be updated to reflect the new size and position. As in (4.1), relaxing this requirement also introduces a feedback loop which is to be avoided for similar reasons.

4.3. Uniform Quantum Size & Aspect Ratio

Initially, all of the quanta (in the test case, photo thumbnails) must be the same size and aspect ratio. Photos that are intrinsically different sizes can be placed on a background that is constant, such that some photos take up the whole background and on some the background shows through on the margins.

Once the layout has been done, this requirement may be removed, so as to minimize wasted space in each region. This needs to be done as a later step to avoid feedback in the initial layout algorithm.

4.4. Secondary Regions Distributed in Quadrants

The algorithms are based on having four fixed quadrants surrounding the primary region. Within each quadrant several secondary region can be placed, but no secondary region can cross a quadrant boundary. Additionally, each quadrant must contain at least one region. The benefit is a tight alignment of secondary regions with the edges of the primary region, thus making a visually appealing layout that enables easy-to-scan grids of photos in each secondary region.

The alternatives are either to tighten this requirement or to remove it entirely. Tightening to eight octants (eight boxes surrounding the primary region) would maintain alignment with the primary region to some degree, but would allow degenerate cases with L-shaped and C-shaped regions, which could pose scanning problems as the viewer tries to determine if the photos should be viewed left-to-right, up-and-down, etc. (Rgn4 in Figure 5b). In the case of quadrants, this is only a problem in <4 regions, and it can be avoided by forcing the primary region to a corner in these cases. For octants, L- and C-shaped regions remain a possibility for up to 8 regions, which cover many of the most common scenarios.

Removing the quadrant requirement entirely complicates things even further, as C-shapes and L-shapes would be allowed with no guarantees of lining edges up with the primary region (Figure 5c). Trying to place regular thumbnails in regions 7 or 5 of Figure 5c could prove difficult. Additionally, these layouts that cause irregularly shaped regions make it difficult to provide an easily scan-able grid of photos.

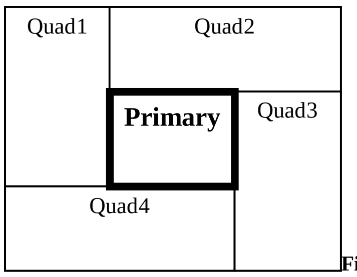


Figure 5a: Four quadrants distributed about a primary region

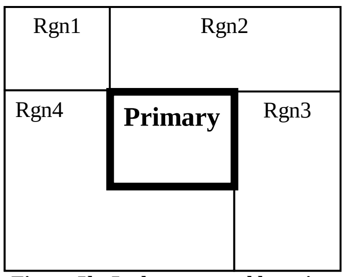


Figure 5b: L-shape caused by using octants for secondary regions

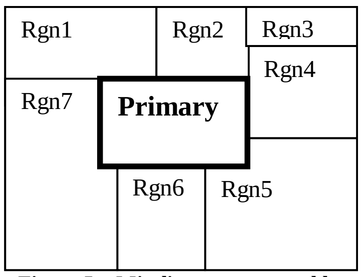


Figure 5c: Misalignments caused by arbitrary region placements

4.5. Fixed Number of Thumbnails

The number of thumbnails in a given region is fixed, in that the algorithm cannot arbitrarily add or remove thumbnails to create a more balanced layout.

If one region has dramatically more thumbnails than another, an application may find it advantageous to allow a region to have only *x* times the number of thumbnails of another, or to specify a minimum thumbnail size which (given a fixed canvas size) necessarily changes the number of thumbnails that are visible. In these cases, a scrollbar or a “More” button could be added to view the thumbnails that were left out.

For the purposes of this discussion, we enable users to decide on the number of thumbnails, but once that decision is made, the number of thumbnails that the algorithm works with is fixed. Allowing the algorithm to internally fine-tune these parameters introduces the undesirable feedback loop discussed in previous sections.

4.6. Fixed Order of Regions

The regions are to be laid out in the order in which they are added. This is useful in applications where order is important, whether it is alphabetic, age-order of children, or page order for a table of contents. Preserving order offers the additional benefit of stability under resize, as regions jumping around when the canvas is resized has been shown to be distracting to users and undesirable in general [4].

5. BI-LEVEL RADIAL QUANTUM LAYOUT

Based on the requirements in Section 4, the following algorithms will generate regions with the largest possible thumbnail size. We describe the layouts as **bi-level** to indicate our solution is for two-level hierarchies, as **radial** to indicate that the secondary regions wrap around the primary region in an ordered manner, and as **quantum** to indicate that the items in each region are fixed in size and shape. Because of these key features we call the algorithm BRQ, pronounced “brick” because of the playful resemblance to a brick wall.

For future work we think three level hierarchies present an even more difficult challenge. If the secondary regions do not have to be wrapped around the primary region, then new possibilities arise, but we do not see useful applications for such layouts. If secondary regions do not have to contain quantized items, then the problem becomes much simpler, but still interesting for some applications.

5.1. QuickLayout Algorithm

The BRQ layout algorithm can be broken up into three steps, which are described here and elaborated on in sections 5.1.1, 5.1.2, and 5.1.3-1.4, respectively:

1. Distribute the secondary tiles among the 4 quadrants, using the QuickLayout algorithm (5.1).
2. Set the initial quantum width/height, which is guaranteed to be an upper bound on the possible quantum dimensions, using the InitialQuantumDim algorithm (5.2).
3. Reduce the quantum dimensions (keeping the same aspect ratio) until there is no overflow, using the ReviseQuantumDim algorithm (5.3). If the quantum dimensions drop below a specified minimum, handle the layout as degenerate (5.4).

5.1.1 QuickLayout Algorithm

The QuickLayout algorithm will optimally divide regions up among the 4 quadrants. It is inspired by the QuickSort algorithm, but instead of splitting a sequence into two groups, it splits it into four groups to fill the four quadrants.

1. Let *total* = the total number of thumbnails in the entire layout
2. Let *rs1* and *rs2* be an array of 2 region sets.
3. Let *clientArea1* and *clientArea2* be the pixel area in each set that is available for thumbnails, excluding borders, text rectangles, etc.
4. Let *areaRatio* = *clientArea1 / (clientArea1 + clientArea2)*
5. Go through each secondary region, adding it to *rs1,* until the number of thumbnails in *rs1 > total* * *areaRatio.*
6. If removing the most recently added region from *rs1* gets the number of thumbnails in *rs1* closer to (*total / 2*), do so.
7. Put the rest of the regions in *rs2.*
8. Enforce the requirement that there be at least one region in each quadrant by requiring at least 2 regions in *rs1* and *rs2* if this is the first call to QuickLayout, and otherwise requiring at least 1 region in *rs1* and *rs2.*
9. If this is the first call to QuickLayout , call QuickLayout (once) recursively on *rs1* and *rs2.*

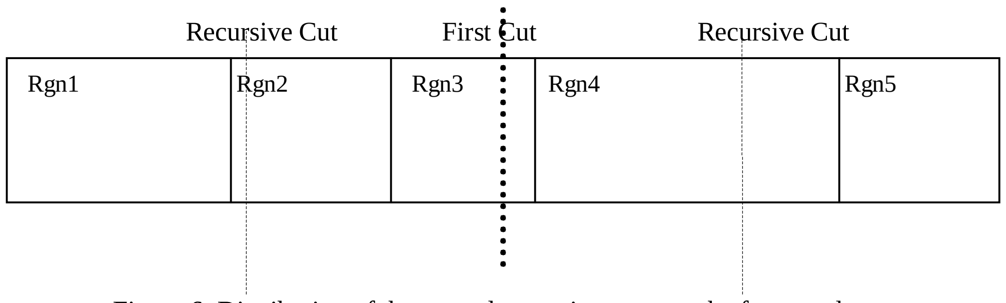


Figure 6: Distribution of the secondary regions among the four quadrants

Figure 6 shows the first three regions would be placed into *rs1*, and the next two would go in *rs2*. On the second recursion, the first would go into Quadrant1, the second and third in Quadrant2, the fourth in Quadrant3, and the fifth in Quadrant4.

The QuickLayout algorithm requires 3 calls, each time looking at SR/2, SR/4, and SR/4 regions respectively, where SR is the number of secondary regions. Thus the time for QuickLayout scales linearly with SR.

5.1.2. InitialQuantumDim Algorithm

The InitialQuantumDim algorithm will set initial dimensions for maximal thumbnail size. This means that the output (*tlength*, *twidth*) of the algorithm is defined to be an upper bound on the size of the thumbnails. In the (rare) case where every region has exactly *rows* x *columns* thumbnails, this will also be the final thumbnail size. In every other case, the thumbnail size will need to be reduced until there is no overflow.

1. Create four rectangles with the same dimensions as the quadrants, as in Figure 7:

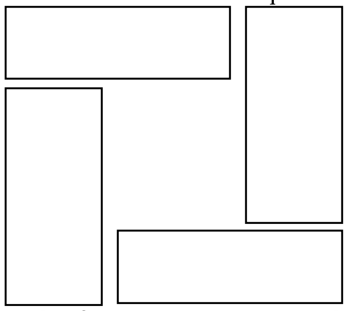


Figure 7: Quadrant rectangles for computing an upper bound on thumbnail size

2. Let $|t_i|$ $\{1 < i <= 4\}$ = the total number of thumbnails in the quadrant by summing the total number of thumbnails in each region in the quadrant.
3. If the thumbnails were to exactly fit the space, then the following would be true for each quadrant:

   $theightCandidate * twidthCandidate * |t_i| = clientArea_i$

4. Solve for quadrant 1, using the known aspect ratio of the thumbnails:

   $theightCandidate = \mathrm{Sqrt}(clientArea_1 * thumbnailAspectRatio / |t_1|)$
   and
   $twidthCandidate = theightCandidate / thumbnailAspectRatio$

   These two values are now upper bounds on the thumbnail size (see section 6 for why).

5. Solve for the other 3 quadrants, reducing *theightCandidate* and *twidthCandidate* if lower values are found.

The InitialQuantumDim algorithm is simply a set of calculations for each of the four regions, and thus operates in constant time.

5.1.3. ReviseQuantumDim Algorithm

The ReviseQuantumDim algorithm takes as its input dimensions which are an upper bound on the possible thumbnail dimensions. It will revise those dimensions downward until it is never the case that there are more columns (quadrants 2, 4) or rows (quadrants 1, 3) than any of the quadrants has room for.

1. For quadrant 1, determine the number of columns used by the first region as follows:

   $columns = \mathrm{Ceiling}(quadrantClientWidth / tWidthCandidate$

   Using the Ceiling allows for columns which are only partially full, but which still take up horizontal space, as in Figure 8:

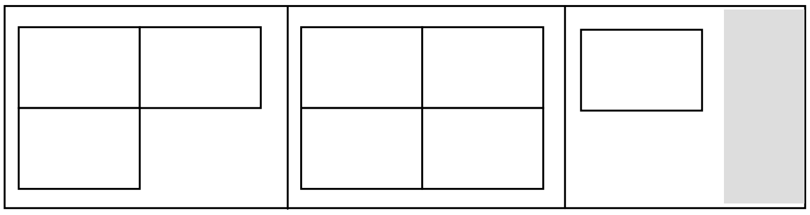


Figure 8: Secondary regions with free space

2. If there is overflow, reduce the thumbnail width by 1 pixel, and repeat Step 1.
3. Now there is no overflow: Redistribute whatever extra space is at the end (shaded in Figure 8) among all of the regions in the quadrant.
4. Using the current thumbnail size, repeat for the other quadrants.
5. When all quadrants have been processed, the last thumbnail size will be the correct one.

The ReviseQuantumDim algorithm starts with the upper bound on the thumbnail width, and reduces by one each time until it reaches either a defined minimum, or zero in the worst case. If it needs to go until 0, the algorithm will run in O(maximum thumbnail width). This factor will only get large in the case of a large display with few thumbnails, and even in that case it will likely terminate before the thumbnail width reaches zero.

5.1.4. Degenerate Layouts

The algorithms described in sections 5.1-5.3 work well for a center rectangle. However, as the rectangle is moved or sized such that a quadrant does not have room for any regions, degenerate layouts occur, as in Figure 9:

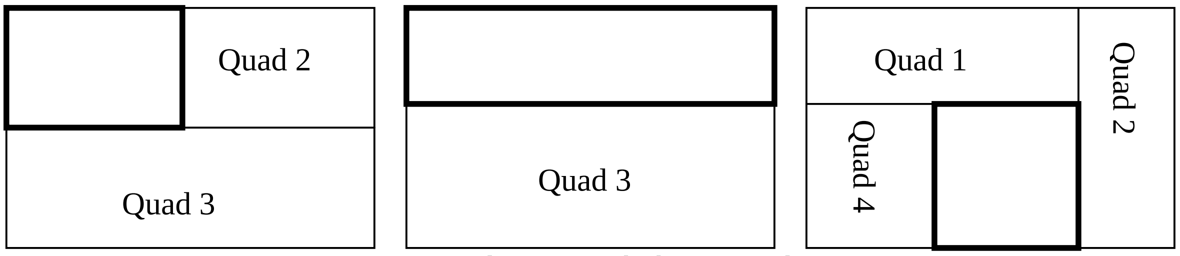


Figure 9: Degenerate layouts, in which some quadrants are empty

To correctly handle these layouts, the QuickLayout algorithm must be modified as follows:

Determine the number of non-empty quadrants:
1. If there are 4, proceed to the normal QuickLayout.
2. If there are 3, proceed to TriQuickLayout
3. If there are 2, proceed to Step 4 of QuickLayout (after the first partition)
4. If there is 1, allocate all regions to that quadrant.
5. If there are 0, do not show any regions.

Each of these layout algorithms runs in time that is linear with the number of regions, as described in the initial QuickLayout section.

TriQuickLayout

This is a special version of QuickLayout to handle division of a collection of n discrete-sized pieces as evenly as possible in thirds:

1. Divide the n regions into two piles, as described in the QuickLayout algorithm. At this point, there are two regions closest to the QuickLayout splitpoint; one to the left and one to the right, at distances a and b:

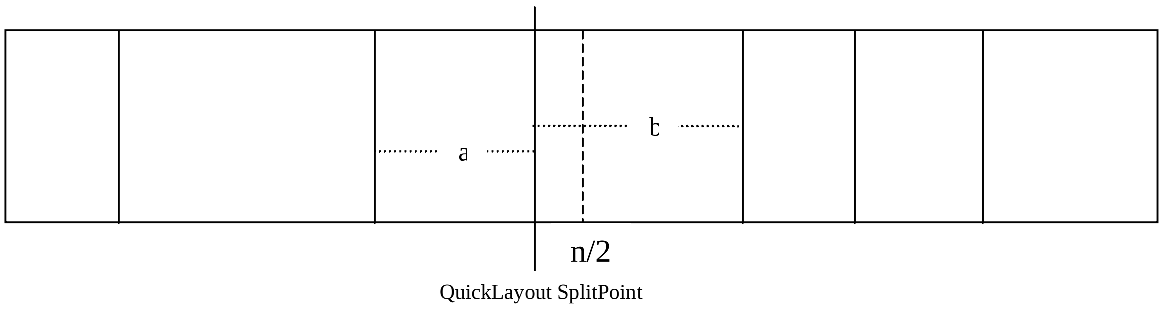


Figure 10: Dividing secondary regions among three "quadrants"

2. Move one of the regions adjacent to the QuickLayout splitpoint to a third pile, such that after the move, the average distance from each side to the absolute center n/2 is as close to equal as possible. In Figure 10, b would be added.
3. Repeat Step 2 until adding a region causes |n/3 - new pile's area| to be more than it was before that iteration of Step 2. When this happens, backtrack one and save the result as the answer.

5.2 Post Processing

Once the algorithm has run, certain actions can be done to incrementally improve the automatically generated layout. We describe two here, although other possibilities exist. In general, these actions can be suggested by going over the requirements of section 4, and violating one of them at this later stage, whereas violating them during the initial layout would cause undesirable feedback and backtracking. For example, violating Requirement 4.2 (Fixed Primary Region Size & Location) suggests resizing or repositioning the primary region to get a "better" layout according to some metric.

5.2.1 Vary Thumbnail Size

There was consistent feedback from the informal user study that users did not like the wasted space frequently generated by the "uniform thumbnail size" requirement. As a result, once the initial algorithm has run, it is possible to increment the thumbnail size in each region until any further increase would cause overflow of the region. This allows each region to have a minimum of wasted space, at the expense of the photos in different regions no longer lining up. This must be done as a post-processing step to avoid violating Requirement 4.3 (uniform thumbnail size).

5.2.2 Add Scrollbar

In order for the layouts to scale properly, a scrollbar can appear in a region if that region has substantially more thumbnails (currently set at 20x) than the smallest region, or than any other region, depending on user preference. The addition of the scrollbar involves deciding how many thumbnails to show (some maximum per region), and then adding to that number on a per-region basis in order to enforce a full grid when not all photos are visible. For example, if a maximum of 40 thumbnails out of a region's 70 are to be shown, and there are 7 columns, the fifth row will have only five thumbnails. In this case, 2 thumbnails should be added to the last row so that the grid will be full unless the scrollbar is at the last position. These must be added as a post-processing step so that Requirement 4.5 (fixed number of thumbnails) is not violated during the initial layout.

5.3 Dynamic Behavior

The algorithms described above allow for interactive, dynamic behavior that encourages experimenting with primary rectangle placement, size, and overall dimensions. This goal led us to develop computed layouts, avoiding the hill-climbing or backtracking strategies that are often used in constraint satisfaction problems. The typical behavior we wished to support is to allow users to move the primary rectangle from the upper left corner into the center, and then enlarge it to highlight its contents (Figure 11). Other behaviors include user resizing of the canvas and addition/deletion of regions as well as thumbnails.

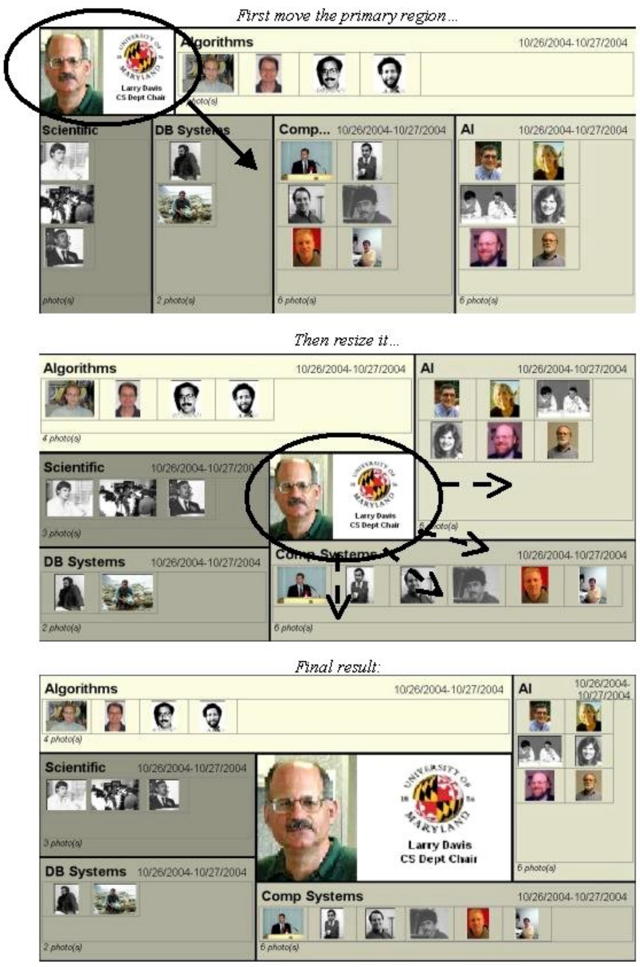


Figure 11: Dynamic Behavior of BRQ-Layout showing user control over dragging and sizing of the primary region

5.4. Performance

To confirm the dynamic behavior described for these algorithms, we conducted 3 trials, in which regions were incrementally added with 10, 50, and 100 thumbnails per region, until there were 100 regions added. Execution time was measured for the three algorithms, demonstrating that the algorithms were rapid enough, with no unforeseen explosions even at the high end of 100 thumbnails in each of 100 regions, for a total of 10,000 quanta being positioned. For this text case, the QuickLayout, InitialQuantumDim, and ReviseQuantumDim algorithms took 0.037ms, 0.014ms, and 0.14ms respectively on a Pentium IV 2.4GHz with 1GB of RAM. Additional tests confirmed the linear growth of time with number of thumbnails.

The drawing of the photos was more time consuming, exerting as much as three orders of magnitude (350ms) greater drag on performance than any of the resizing algorithms. With better image storage allocation, these photo display problems can be solved, independent of the algorithmic performance outlined in this paper.

6. OPTIMALITY DISCUSSION

It is hard to quantify what makes one photo layout “better” than another, as subjective measures could vary in different applications and for different users. For our purposes, however, we will use the following definition to try to describe an “optimal” layout:

A layout is said to be optimal if there is no layout that results in larger thumbnails.

Since any available space should be used for increasing thumbnail size, this is equivalent to a second definition:

A layout is said to be optimal if there is a minimum of unused space.

Using these definitions, the algorithms described here produce layouts that are optimal given the requirements in Section 4. The relaxation of one or more of these requirements could provide a layout that better meets these optimality criteria, but at the expense of ease of scanning, consistent region placement, equal thumbnail prominence, and other problems discussed in Section 4. The following sections discuss the optimality of algorithms 5.1, 5.2, and 5.3 respectively.

6.1. Distribution of Regions into Quadrants

QuickLayout divides the regions among the quadrants as evenly as possible. Imbalances tend to occur for a thumbnail distribution which is very heavy on one side (for example: 5 regions, where 1-4 have 5 thumbnails each and region 5 has 50). In this case, QuickLayout wants to put regions 1-4 in quadrants 1 and 2, but then region 5 would need to get placed in quadrants 3 and 4, violating requirement 4.4 and causing the undesirable L-shaped region. So in this example, regions 4 and 5 would go into quadrants 3 and 4, causing a reduction in thumbnail size mandated by requirement 4.4.

6.2. Upper Bound on Thumbnail Size

The InitialQuantumDim algorithm calculates a value for thumbnail size using height, width, and number of thumbnails, ignoring row vs. column layouts and uneven distribution among regions. To show that this is an upper bound, consider w', a proposed thumbnail width that is wider than w, the value claimed to be maximal. (Since the aspect ratio is fixed, this also implies an h' > h.) Therefore, w' x h' x (# thumbnails) > $clientArea_i$ as defined in 5.2, which means that the thumbnails take up more absolute area than is available. Thus the values w and h are upper bounds from which ReviseQuantumDim can confidently revise only downward.

6.3. Final Thumbnail Size

The ReviseQuantumDim algorithm will always result in at least one quadrant containing all of its rows (quadrants 2, 4) or columns (quadrants 1, 3) having at least one thumbnail, thus tightly fitting the set of thumbnails to the quadrant's client area. As a result, the other quadrants by definition have a minimum of unused space and the largest possible thumbnails, since if the thumbnails were made any larger they would overflow the tightest fitting quadrant (because all thumbnails are required to be the same size).

6.4. Improvements by Relaxing Requirements

There are several ways in which users may want to improve the quality of the generated layout, by manually adjusting some of the requirements in Section 4. Users could choose, for example, to increase the primary rectangle's dimensions, or change its position, if an initial layout showed extra space. A manual change in ordering of the regions could also result in a better layout. The "same size thumbnails" requirement might be relaxed, with thumbnails growing to different sizes in their various regions until there is no wasted space, at the expense of having different sized thumbnails in each region.

Each of these user actions should be viewed as being at the application level: The algorithms discussed here provide a starting point given certain requirements, and if those are modified, they will again generate a best layout based on the new requirements. Any interaction between the application level and the algorithm layer causes feedback loops that make deterministic solutions impossible.

If a better solution were desired, at the expense of the feedback loops described above, an approach would be to generate a set of thumbnail dimensions for points *about* the current solution, for a given requirement. For example, once the algorithm has finished, it could generate "what-if" scenarios, varying the size of the main rectangle or its position by up to some threshold. Depending on how many requirements were violated and to what extent, some number of hypothetical dimensions could be generated. Since the calculations are relatively trivial, many of these could be generated in a short period of time, and by looking at this space, a better solution could be found. This type of operation should only be allowed at the explicit request of the user, as someone trying to exactly locate the primary rectangle could be frustrated by an automated agent "improving" the position to enlarge the thumbnails by changing the position that the user is trying to obtain.

6.5. User Evaluation

To gauge user responses to these bi-level radial quantum layouts we asked four knowledgeable users of photo library software to review our interface for 30-40 minutes each [8]. In this modest usability study, they were shown the on-screen, but static layouts in Figures 1-4, in order and asked what they understood about the layout and relationship among the regions in each Figure. The users understood why a particular thumbnail size was chosen, although sometimes only after carefully comparing regions to find the one that was constraining. Several voiced interest in a feature that would relax the "same thumbnail size" requirement in order to have less wasted space. They all appreciated the ability to manipulate the layout in real time.

7. CONCLUSION

This paper has described requirements and the BRQ layout algorithm to layout a large collection of items in a bi-level hierarchy. Many applications could use these algorithms to do dynamic layout quickly and deterministically, and digital photo layouts are an especially important example of such an application. A vital aspect of the BRQ layout algorithm is its rapid performance, which enables compelling interactive experiences as users resize the canvas, add/delete regions, or add/delete items to a region.

Future work includes investigation of the requirements described here, with an eye to removing some of them and employing hill-climbing, backtracking, or other techniques to circumvent non-linearities in the resulting equations. A further challenge is extend these ideas to a three-level hierarchical layout, which presents additional difficulties. A three-level layout of digital photos could be useful to show grandparents, parents and grandchildren in one large ensemble, or an organization chart with COO, VPs & senior managers in concentric rectangles.

8. ACKNOWLEDGEMENTS

We would like to thank Adobe Corporation for their support, which has made this research possible. We would also like to thank Ben Bederson, Alexander Loui, and Martin Wattenberg and for their valuable comments. We would also like to thank Jonathan Katz for his suggestions regarding the thumbnail sizing algorithms.

9. REFERENCES

[1] ACDSystems, http://www.acdsystems.com/English/Products/ACDSee/index.htm. Last visited February 15, 2005.

[2] Adobe Photoshop Album, http://www.adobe.com/products/photoshopalbum/main.html. Last visited February 15, 2005.

[3] Apple iPhoto, http://www.apple.com/ilife/iphoto/. Last visited February 28, 2005.

[4] B. Bederson, B. Shneiderman, and M. Wattenberg (2002). Ordered and quantum treemaps: Making effective use of 2D space to display hierarchies, *ACM Transactions on Graphics 21*, 4, 833-854.

[5] J. Geigel and A. Loui (2003). Using genetic algorithms for album page layouts. *IEEE Multimedia 10*, 4, Oct-Dec 2003,16-27.

[6] C. Jacobs, W. Li, E. Schrier, D. Bargeron, and D. Salesin (2003). Adaptive grid-based document layout. *ACM Transactions on Graphics*, 838--847, July 2003. ACM Press, New York.

[7] B. Johnson and B. Shneiderman (1991). Treemaps: a space-filling approach to the visualization of hierarchical information structures. *Proc. 2nd International IEEE Visualization Conference*, 284-291. IEEE Computer Society, Washington DC.

[8] J. Kustanowitz. and B. Shneiderman (2005). Meaningful presentations of photo libraries: Rationale and applications of bi-level radial quantum layouts, *Proc. Joint Conference on Digital Libraries*, ACM Press, New York, (to appear, June 2005).

[9] PhotoMesa, http://www.photomesa.com Last visited February 15, 2005.

[10] Picasa, http://www.picasa.com/picasa/. Last visited February 15, 2005.

[11] L. Purvis, S. Harrington, B. O'Sullivan, and E. C. Freuder (2003). Creating personalized documents: an optimization approach. *Proceedings of the ACM Symposium on Document Engineering*, 68-77. ACM Press, New York.